❑Create pods, deployments, services.

Answere-->

1. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.It is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

Creating the pods-->

```
apiVersion: v1
kind: Pod
metadata:
        name: nginx
spec:
        containers:
        - name: nginx
                image: nginx:1.14.2
                ports:
                - containerPort: 80
```

To create the Pod shown above, run the following command-->

```
kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
```

2. A Deployment provides declarative updates for Pods and ReplicaSets.Provide a YAML example of a deployment configuration

Yaml Example-->

```
apiVersion: apps/v1
kind: Deployment
metadata:
        name: nginx-deployment
spec:
        replicas: 3
        selector:
                matchLabels:
                        app: nginx
        template:
                metadata:
                        labels:
                                        app: nginx
                spec:
                        containers:
                                        - name: nginx-container
                                        image: nginx:latest
```

use this command-- kubectl apply -f nginx-deployment.yaml

3. In Kubernetes, a Service is a method for exposing a network application that is running as one or more

Pods in your cluster.

```
                    apiVersion: v1
                    kind: Service
                    metadata:
                              name: nginx-service
                    spec:
                              selector:
                                        app: nginx
                              ports:
                                        - protocol: TCP
                                        port: 80
                                          targetPort: 80
```

use this command-->kubectl apply -f nginx-service.yaml

2   Deploy pods on specific node using concept of taint & tolerance, node selector & node affinity

Taint allow a node to repel a set of pods.Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes.

```
          add tolarance kubectl taint nodes node1 key1=value1:NoSchedule
          tolerations:
          - key: "key1"
                    operator: "Equal"
                    value: "value1"
                    effect: "NoSchedule"
```

Node selector is a straightforward way to constrain which nodes your pod is eligible to be scheduled on based on labels.

```
          spec:
                    nodeSelector:
                              <label-key>: <label-value>
```

Node affinity provides more advanced expressions to control pod placement based on node properties such as labels.

Here's a complete example that combines taints and tolerations, node selector, and node affinity

```
                    apiVersion: v1
```

```
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: myapp
    image: myapp-image
  nodeSelector:
    disktype: ssd
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: machine-type
            operator: In
            values:
            - large
  tolerations:
  - key: "specialnode"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

3 Explore node affinity, node selector, anti affinity

Answere-->

node selector-->

Node selector is the simplest form of influencing pod scheduling based on node labels. You specify a set of key-value pairs in the pod's specification (spec.nodeSelector) that the node must have in order for the pod to be scheduled on it.

```
spec:
        containers:
                - name: nginx
                image: nginx
          nodeSelector:
                disktype: ssd
```

This YAML instructs Kubernetes to schedule the pod only on nodes that have the label disktype=ssd.

Node Affinity-->

Node affinity allows for more complex rules than node selector. It specifies rules that govern how pods are scheduled onto nodes based on the labels of the node.

```
spec:
  containers:
    - name: nginx
      image: nginx
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
```

This YAML specifies that the pod should be scheduled on nodes that have the label disktype=ssd.

Node anti-affinity -->
specifies rules that prevent pods from being scheduled onto nodes that match certain conditions, such as node labels or topology constraints.

```
spec:
  containers:
    - name: nginx
      image: nginx
  affinity:
    nodeAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
```

This YAML specifies that the pod should avoid nodes that have the label node-role.kubernetes.io/master

4.. Deploy mysql pod and store the data in a persistent volume. Delete the pod and attach the volume to a new pod. Check if the data is available to the new pod.

create yaml and apply pvc ----

```
kubectl apply -f mysql-pvc.yaml
```

deploy sql pod---

```
kubectl apply -f mysql-deployment.yaml
```

Access the MySQL Pod and insert some data:

```
kubectl exec -it <mysql-pod-name> -- mysql -u root -p
```

delete my sql pod---

```
kubectl delete pod <mysql-pod-name>
```

Attach the PVC to a New Pod--

```
kubectl apply -f mysql-new-pod.yaml
```

Access the new MySQL Pod and check if the data inserted earlier is still there

```
kubectl exec -it mysql-new -- mysql -u root -p
```

5. Deploy Wordpress to kubernetes and expose the service to outside world using nodeport, loadbalancerservices.

Answere-->

WordPress requires storage for storing uploads and other data. Create a PVC YAML

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wordpress-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
kubectl apply -f wordpress-pvc.yaml
```

Apply the mysql Deployment-- `kubectl apply -f mysql-deployment.yaml`

Apply the WordPress Deployment:-- `kubectl apply -f wordpress-deployment.yaml`

Create a NodePort Service YAML (wordpress-nodeport.yaml):

```
kubectl apply -f wordpress-nodeport.yaml
```

Create a LoadBalancer Service YAML (wordpress-lb.yaml):

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-lb
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: wordpress
```

Apply the LoadBalancer Service: kubectl apply -f wordpress-lb.yaml

The LoadBalancer Service will provision an external IP (on supported cloud providers) to access WordPress.
For NodePort: Access WordPress using any node's IP address and the NodePort
For LoadBalancer: Access WordPress using the external IP provided by your cloud provider.

6.. Define resource limit for containers in the wordpress deployment.
Answere--->
When deploying WordPress on Kubernetes, it's essential to define resource limits for containers to ensure efficient resource utilization and prevent resource contention. Resource limits specify the maximum amount of CPU and memory that a container can use.
it specifies the maximum amount of CPU and memory that the container can use.

7.. Define Readiness & liveness probes for the wordpress container.
Answere-->
Readiness Probe--->
The readiness probe determines when a container is ready to start accepting traffic. If a readiness probe fails, the Kubernetes Service removes the Pod's IP address from its set of endpoints, meaning it won't receive traffic until it passes the readiness probe again.

Liveness Probe--->
The liveness probe checks if the container is still alive and responsive. If a liveness probe fails, Kubernetes restarts the container to restore it to a healthy state.